

Cursos, soluções e serviços baseados em software livres e padrões abertos para ambientes de missão crítica

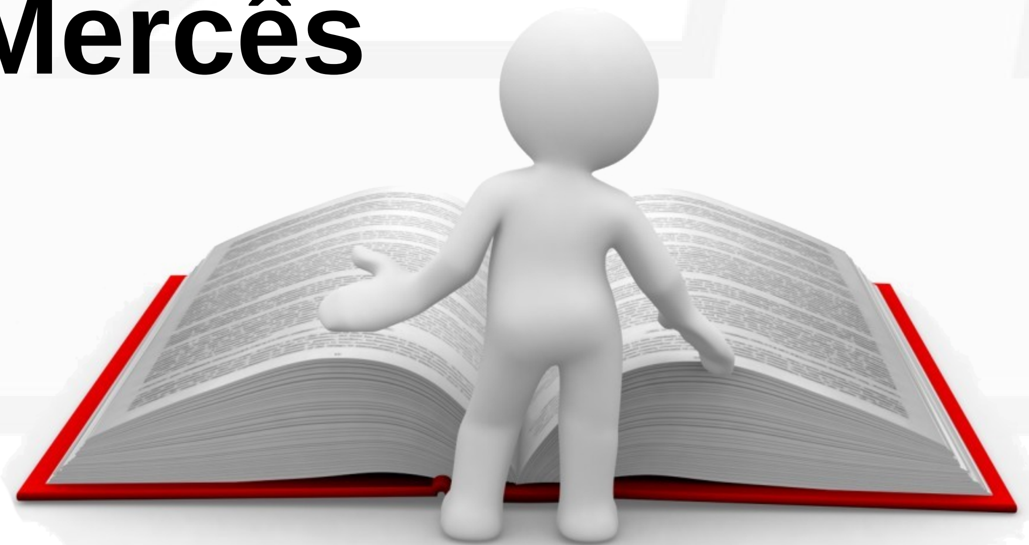
- *Experiência em missão crítica de missão crítica*
- *Pioneira no ensino de Linux à distância*
- *Parceira de treinamento IBM*
- *Primeira com LPI no Brasil*
- *+ de 30.000 alunos satisfeitos*
- *Reconhecimento internacional*
- *Inovação com Hackerteen e Boteconet*



4LINUX
FREE SOFTWARE SOLUTIONS

Análise de malware com Software Livre

Fernando Mercês



No final da minha palestra



➔ **As 05 primeiras perguntas ganharão um botton do Tux.**

➔ **Sorteio do curso "Investigação Forense Digital – 427 (EAD)"**

• Preencha o cupom que está junto ao folheto que você recebeu na entrada da palestra;

• Se você já preencheu, ele já está aqui na urna.

• O ganhador deve estar presente até o quinto sorteio. Se não estiver presente, ganhará o sexto sorteado.

Análise de malware

4LINUX
FREE SOFTWARE SOLUTIONS

➔ O que é um malware?

* “Malicious software”, é um software indejesado geralmente com intenções nada nobres como roubo de informações, replicação, comprometimento do sistema etc.

➔ O que é análise de malware?

* É a técnica de documentar o comportamento de um software suspeito sem seu código-fonte.

Malware no Linux existe?

4LINUX
FREE SOFTWARE SOLUTIONS

Winter, Winux/Lindose, Diesel, Coin, Mighty, Adore, Slapper, Kork, Millen... [1]

- Existem vários malwares para o sistema do pinguim, mas a quantia não é realmente grande se comparada à quantidade de malwares para o Janelas.

- É bom se preparar enquanto é tempo. :)

Agenda

4LINUX
FREE SOFTWARE SOLUTIONS

1 - Análise estática e reconhecimento do local
- **1.2** - Who are you? Com a ferramenta file
- **1.3** - Busca abusada de strings
- **1.4** - Raio-x com o objdump

2 - Binário protegido, e agora?
- **2.1** - Exemplo de (des)compactação com packer livre
- **2.2** - Outras proteções possíveis

3 - Rastreando os passos
- **3.3** - Começando do começo
- **3.2** - Entendendo as calls e as syscalls
- **3.3** - Documentação do comportamento (ou "te peguei")

Análise estática e reconhecimento do local

Who are you?

4LINUX
FREE SOFTWARE SOLUTIONS

- A maioria dos tipos de arquivo possui um **“magic number”** para identificação, localizado logo nos primeiros bytes.

- A ferramenta file [2] está presente em todas as distribuições GNU/Linux. Ela utiliza a biblioteca libmagic como “core” para reconhecer tipos de arquivo.

```
$ file /etc/hosts
/etc/hosts: ASCII English text

$ file /bin/ls
/bin/ls: ELF 64-bit LSB executable, x86-64,
version 1 (SYSV), dynamically linked (uses
shared libs), for GNU/Linux 2.6.18, stripped
```

Who are you?

4LINUX
FREE SOFTWARE SOLUTIONS

• O file não se limita a identificar o tipo de arquivo, mas dá várias informações sobre alguns tipos, como é o caso de binários ELF.

• Script do Nautilus “Get info” para agilizar o reconhecimento:

```
#!/bin/bash
info=`file "$1"`
ls=`ls -l "$1"`
wc=`wc "$1"`
size=`ls -lh "$1" | cut -d" " -f5`

zenity --info --text="\$ file\n$info\n\n\$
wc\n$wc\n\n\$ ls\n$ls" --title="$1 ($size)"
```

Busca abusada de strings

4LINUX
FREE SOFTWARE SOLUTIONS

• Graças ao conjunto binutils [3], temos uma ferramenta livre chamada strings, que nos exibe todos os textos dentro de um arquivo qualquer, inclusive binários.

```
$ strings /bin/ls | tail
%H:%M:%S
%m/%d/%y
%Y-%m-%d
invalid suffix in %s%s argument `%s'
invalid %s%s argument `%s'
%s%s argument `%s' too large
xstrtol.c
0 <= strtol_base && strtol_base <= 36
xstrtoul
xstrtoumax
```

Busca abusada de strings

4LINUX
FREE SOFTWARE SOLUTIONS

• A análise de strings é extremamente importante. Nela podem constar informações essenciais para o malware como nomes DNS, caminhos e nomes de diretórios e arquivos etc.

• Conhecendo esta "falha", os criadores de malware frequentemente encriptam as strings, de forma que uma análise estática não ajuda muito:

```
$ strings -a -t x binary.exe | tail -4
  180b6 waG(x
  180f6 w52Wx
  18196 vvWx
  18314 ~Ftx
```

• Veremos como desencriptar essas strings mais à frente.

Raio-X com o objdump

4LINUX
FREE SOFTWARE SOLUTIONS

- O pacote binutils contém vários aplicativos interessantes para a análise de malware, dentre eles:

- **objdump**
- objcopy
- nm
- readelf
- size
- strings
- strip

- O objdump é uma ferramenta poderosa, que exibe muitas informações sobre o binário, como veremos a seguir. As outras ferramentas também são muito úteis e ficam como lição de casa. ;-)

Raio-X com o objdump

4LINUX FREE SOFTWARE SOLUTIONS

```
$ objdump -d malware | grep -A30 main

0000000000400587 <main>:
  400587:  push   %rbp
  400588:  mov    %rsp,%rbp
  40058b:  sub    $0x10,%rsp
  40058f:  movl   $0x2823326e,-0x10(%rbp)
  400596:  movl   $0x27286e2f,-0xc(%rbp)
  40059d:  movl   $0x272f2e22,-0x8(%rbp)
  4005a4:  movw   $0x2628,-0x4(%rbp)
  4005aa:  movb   $0x0,-0x2(%rbp)
  4005ae:  lea    -0x10(%rbp),%rax
  4005b2:  mov    %rax,%rdi
  4005b5:  callq  400534 <decrypt>
  4005ba:  lea    -0x10(%rbp),%rax
  4005be:  mov    %rax,%rdi
  4005c1:  callq  400438 <remove@plt>
  4005c6:  mov    $0x0,%eax
  4005cb:  leaveq
  4005cc:  retq
```

Raio-X com o objdump

4LINUX
FREE SOFTWARE SOLUTIONS

No slide anterior vemos um *disassembly* da função *main()* com o objdump. Aqui é importante notar:

• Os nomes das funções foram identificados, então estão no binário compilado. São os *symbols*.

• O *disassembly* é estático. O código não rodou.

• O grep na main foi proposital, mas a execução de um binário não começa na main. Atente para a saída do objdump sem o grep e você verá que tem muito código antes da main ser chamada.

• O objdump utliza sintaxe AT&T no *disassembly*.

Sintaxe Intel com o gdb

4LINUX
FREE SOFTWARE SOLUTIONS

```
$ gdb -q malware
(gdb) set disassembly-flavor intel
(gdb) disassemble main
   0x400587 push rbp
   0x400588 mov rbp,rsp
   0x40058b sub rsp,0x10
   0x40058f mov DWORD PTR [rbp-0x10],0x2823326e
   0x400596 mov DWORD PTR [rbp-0xc],0x27286e2f
   0x40059d mov DWORD PTR [rbp-0x8],0x272f2e22
   0x4005a4 mov WORD PTR [rbp-0x4],0x2628
   0x4005aa mov BYTE PTR [rbp-0x2],0x0
   0x4005ae lea rax,[rbp-0x10]
   0x4005b2 mov rdi,rax
   0x4005b5 call 0x400534 <decrypt>
   0x4005ba lea rax,[rbp-0x10]
   0x4005be mov rdi,rax
   0x4005c1 call 0x400438 <remove@plt>
   0x4005c6 mov eax,0x0
   0x4005cb leave
   0x4005cc ret
```

Binário protegido, e agora?

Packer livre: UPX

4LINUX
FREE SOFTWARE SOLUTIONS

Existem muitas formas de proteger o binário de olhares curiosos e é claro que o criador de malware não quer ninguém bisbilhotando seu arquivo, pois este poderá descobrir como ele funciona, criar uma vacina, denunciar para onde os dados são enviados etc.

Uma proteção bem comum é o packer, que comprime um executável e dificulta muito a análise. O único packer de código aberto que conheço é o UPX [4], por isso vou utilizá-lo em nosso malware.

```
$ objdump -d malware_packed

malware_packed:     file format elf64-x86-64
```

Packer livre: UPX

4LINUX FREE SOFTWARE SOLUTIONS

Como o UPX é um packer open source, para removê-lo basta usar o próprio comando upx:

```
$ upx -d malware
```

Agora o objdump e outras ferramentas de análise já conseguem exibir as informações que buscamos pois o executável está em seu estado original.

Outras proteções

4LINUX FREE SOFTWARE SOLUTIONS

Além dos packers, existe uma série de técnicas para proteção de binários, dentre elas:

- Crypters.
- Obfuscação.
- Dummy code.
- Virtualização.
- CRC.

Até o presente momento, ferramentas para Linux mais rebuscadas para proteção de binários não são comumente distribuídas, mas não se sabe o que há de ferramentas fechadas/individuais por aí. ;-)

Rastreando os passos

Começando do começo

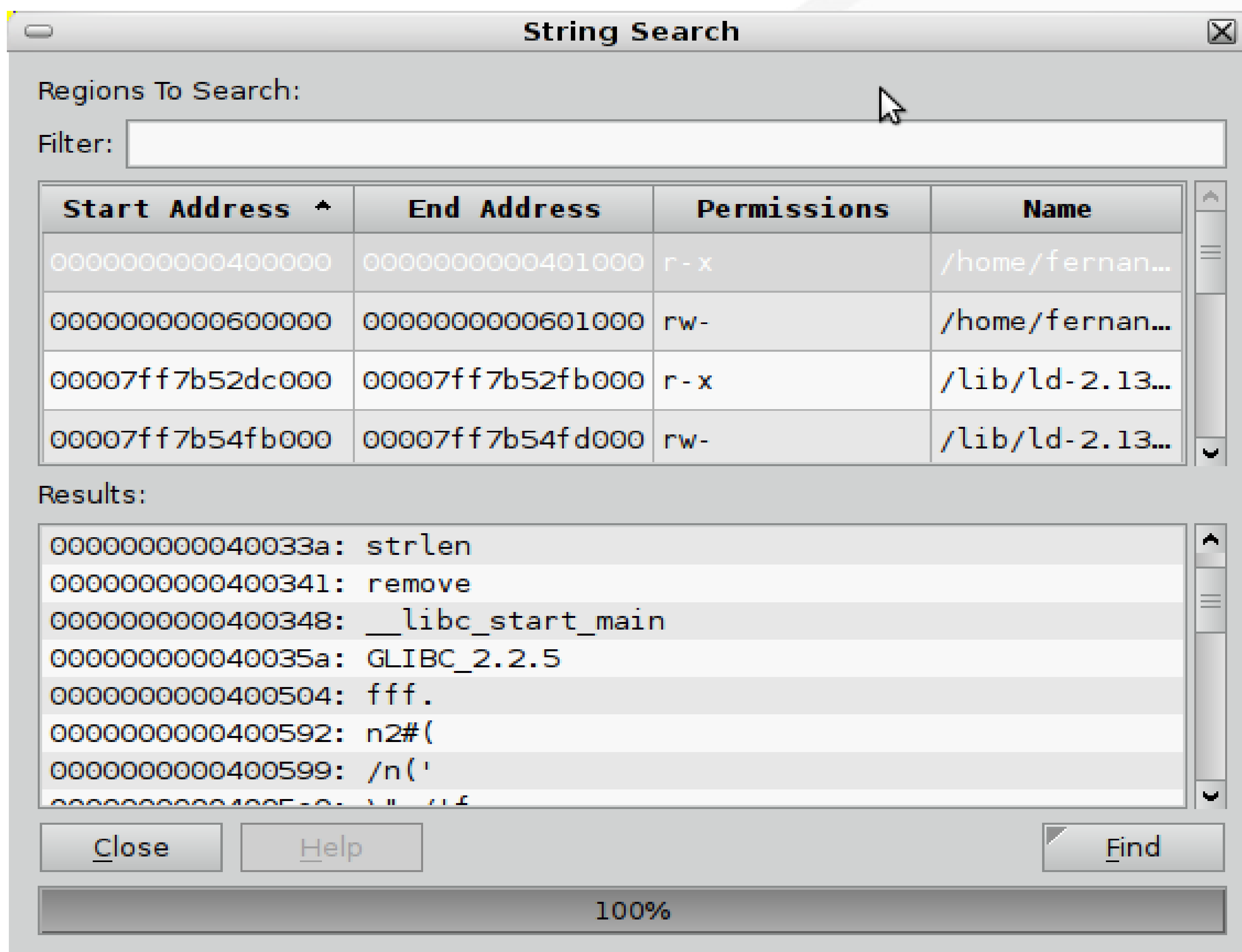
4LINUX
FREE SOFTWARE SOLUTIONS

Aqui abordaremos a análise dinâmica, ou seja, vamos rodar o malware, passo-a-passo, num ambiente controlado, para identificar seu comportamento. Para isso, será usado um poderoso debugger e disassembler livre, o EDB [5].

- Ao abrir o malware no EDB, paramos em seu início (não é a main, lembra?).

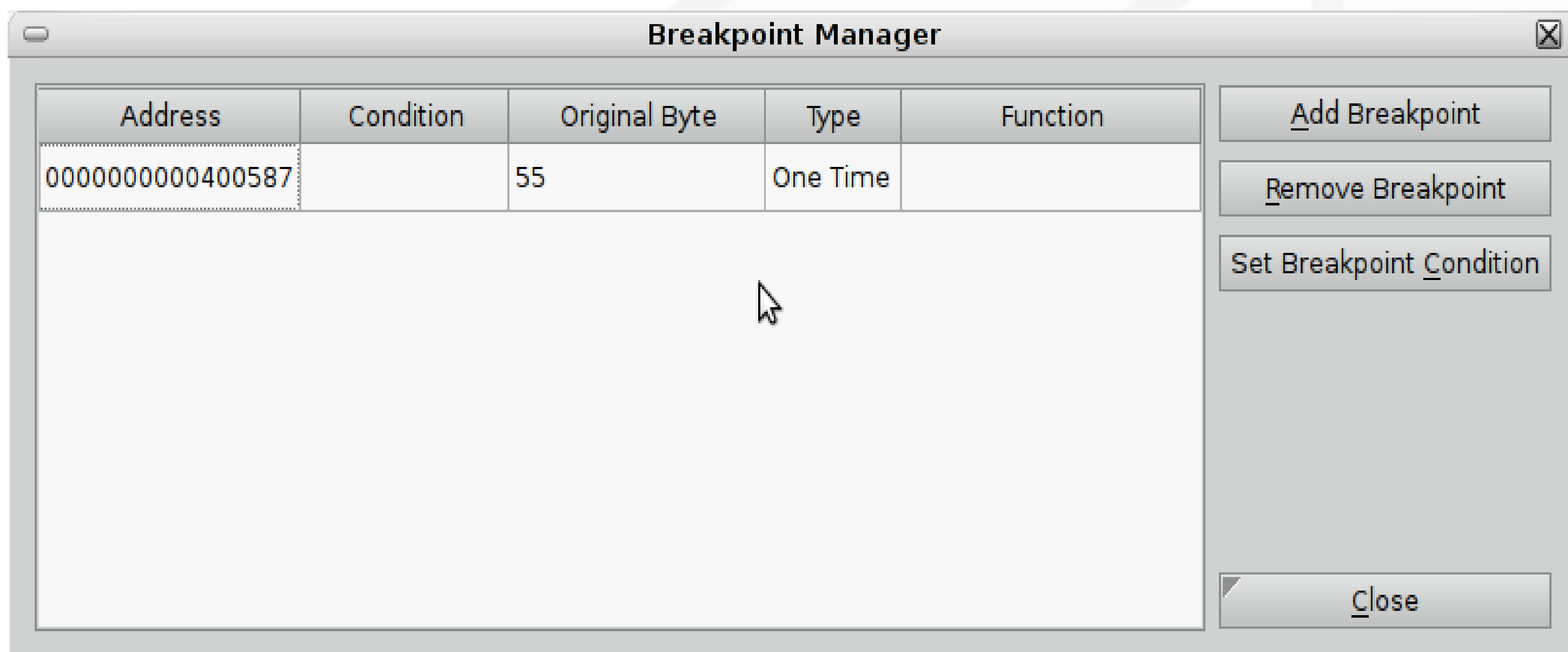
- Podemos buscar strings direto pelo EDB, caso ainda não tenhamos feito com o strings. Para isso, basta um CTRL+S, que vai carregar o plugin String Search.

Começando do começo

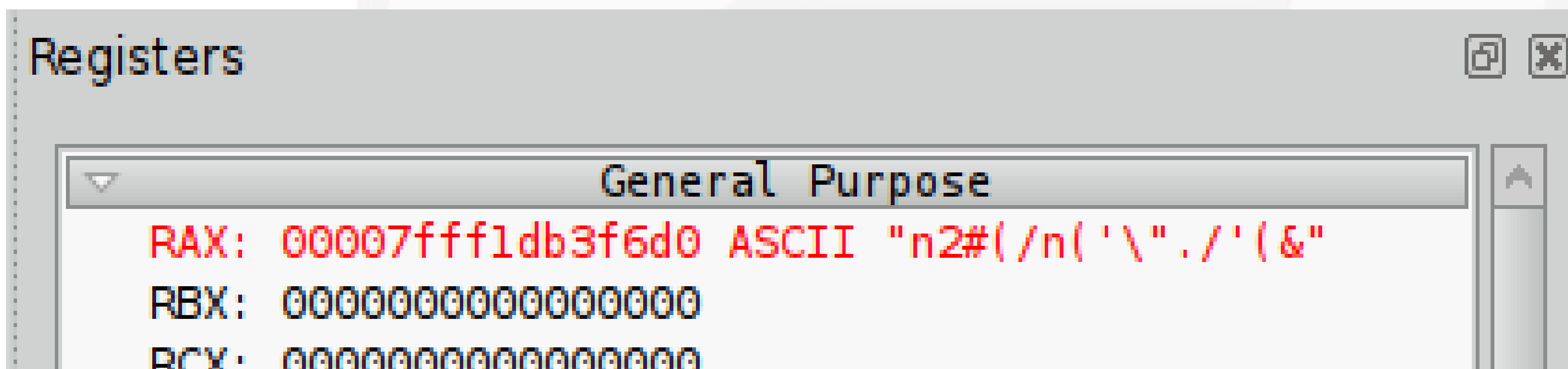


Entendendo as calls

O EDB já coloca um breakpoint na main automaticamente. Para conferir, veja a tela do Breakpoint Manager (CTRL+B):



Entendendo as calls



Podemos mandar rodar o malware no EDB (F9) e aguardar parar no início da função principal então. Isto você pode fazer sem medo, não?

Podemos executar linha a linha com o F8 e em 0x4005ae, vemos que o endereço de uma string interessante é armazenada em RAX:

Entendendo as calls

4LINUX
FREE SOFTWARE SOLUTIONS

Logo abaixo temos uma bendita CALL. Mas o que é isso?

• Qualquer chamada para função gera uma CALL em Assembly.

• O código é desviado para o endereço apontado pea CALL e, após um RET, volta para a instrução logo abaixo à chamada da CALL.

• Os parâmetros de função são passados para a CALL através da pilha (stack).

• O retorno pode variar, mais geralmente é dado em RAX.

Entendendo as calls

4LINUX
FREE SOFTWARE SOLUTIONS

00000000:004005ae	48 8d 45 f0	lea rax, [rbp-16]
00000000:004005b2	48 89 c7	mov rdi, rax
00000000:004005b5	e8 7a ff ff ff	call 0x0000000000400534
00000000:004005ba	48 8d 45 f0	lea rax, [rbp-16]

• A execução será desviada para a instrução em 0x400534 e os parâmetros podem ser vistos na janela "Stack" do EDB.

Stack

00007fff:6051aef0	27286e2f2823326e	n2#(/n('	
00007fff:6051aef8	00002628272f2e22	"./'(&..	
00007fff:6051af00	0000000000000000	....	
00007fff:6051af08	00007fd4c2c7eead	-îÇÂÖ...	return to 00007fd4c2c7eead <libc-2.13.so::__

syscalls

FREE SOFTWARE SOLUTIONS

Já as sycalls são chamadas às rotinas providas pelo SO, por exemplo, para imprimir uma string na tela.

• Alguns exemplos são: setuid(), exit() e fork().

• Podem ser utilizadas diretamente pelo malware, já que este ficará preso à arquitetura (não é comum malware portável :).

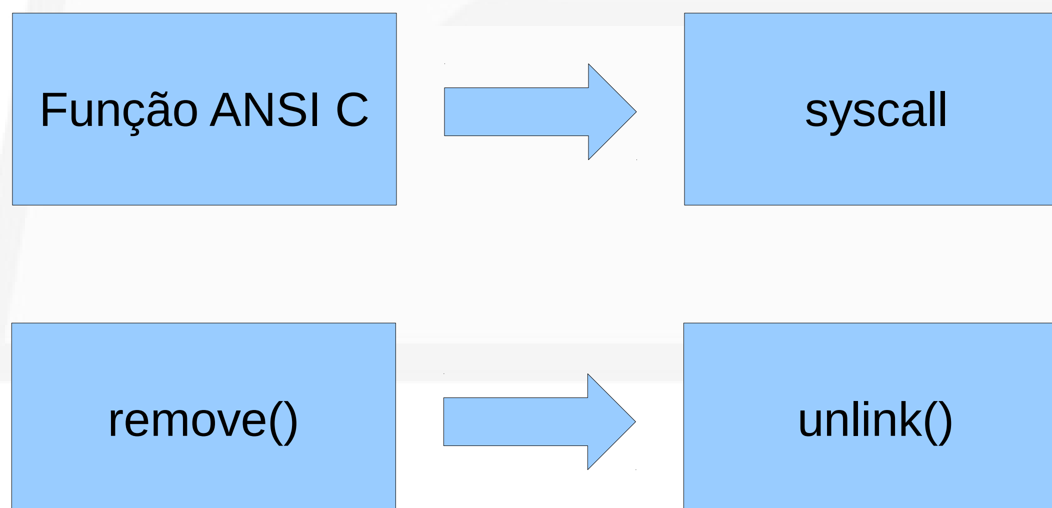
• Existem uma tabela do tipo QR-card com todas as syscalls do Linux [6]. O site original [7] estava for a dor ar (403) no dia em que escrevi este slide.

syscalls

- Quando o binário não tem os symbols, o objdump, readelf, hte e nem mesmo o gdb são capazes de exibir o nome da função em questão, o que é um problema. Já o EDB, consegue pois é possível gerar symbols para cada biblioteca presente.

- Importante notar que funções ANSI C são wrappers para syscalls.

Exemplo:



Documentação

4LINUX FREE SOFTWARE SOLUTIONS

Documentar o comportamento de um binário é um trabalho demorado, porém de máxima utilidade. As vantagens são:

- *Rollback* de ações.
- Criação de vacinas.
- Aprendizado.
- Massagem no ego. :)

Oportunidades

4LINUX
FREE SOFTWARE SOLUTIONS

- Empresas de anti-vírus.
- Grupos de resposta a incidentes.
- Órgãos públicos e militares.
- Times de segurança de empresas privadas.

Referências

4LINUX
FREE SOFTWARE SOLUTIONS

[1] http://en.wikipedia.org/wiki/Linux_malware
[2] http://www.darwinsys.com/file/
[3] http://www.gnu.org/software/binutils/
[4] http://upx.sourceforge.net
[5] www.codef00.com/projects#debugger
[6] http://www.mentebinaria.com.br/files/syscall-table.pdf
[7] http://www.bigfoot.com/~jialong_he

Fonte do pseudo-malware

4LINUX FREE SOFTWARE SOLUTIONS

```
void decrypt(char s[])
{
    int i;

    for (i=0; i<strlen(s); i++)
        s[i] ^= 0x41;
}

int main()
{
    int i, j;

    /* Tenta remover "/sbin/ifconfig" */
    char arq[] = "n2#(/n('\"./'(&";
    decrypt(arq);
    remove(arq);

    return 0;
}
```

NOTA: Para o UPX comprimir esse código foi acrescido com várias funções inúteis.

Referências

4LINUX FREE SOFTWARE SOLUTIONS

[1] http://en.wikipedia.org/wiki/Linux_malware
[2] http://www.darwinsys.com/file/
[3] http://www.gnu.org/software/binutils/
[4] http://upx.sourceforge.net
[5] www.codef00.com/projects#debugger
[6] http://www.mentebinaria.com.br/files/syscall-table.pdf
[7] http://www.bigfoot.com/~jialong_he

Perguntas?

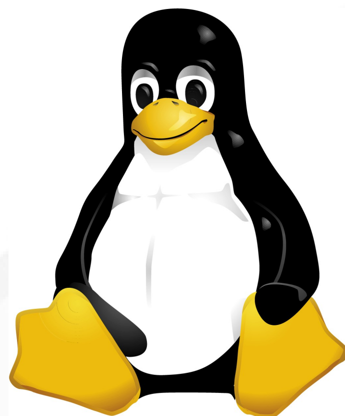


@MenteBinaria

www.mentebinaria.com.br

Muito obrigado!

4LINUX
FREE SOFTWARE SOLUTIONS



fernando.merces@4linux.com.br
www.4linux.com.br
www.hackerteen.com
twitter.com/4LinuxBR

Tel: 55-11-2125-4747